

Package: WFC (via r-universe)

September 14, 2026

Title Workflow-Oriented Survey Weight Calibration

Version 2.0.1

Author Kunxiang Ma [aut, cre], WEIAN DATA TECH (Beijing) Co., Ltd.
[cph, fnd]

Maintainer Kunxiang Ma <makunxiang@weiansdata.com>

Description Provides a disciplined precheck, execution, and diagnostics workflow for survey weighting and raking. Weight construction requires design-only data, a verified external target, outcome-blind planning, and human approval before weight locking, with bilingual reports for decision and statistical audiences. Converts calibrated and replicate weights into standard survey designs, provides optional broom-style result projections, and records serializable production pipeline provenance. Supports fixed, predeclared soft calibration tolerances and categorical entropy balancing from verified margins, plus panel attrition weighting, high-influence unit diagnostics, Fay's balanced repeated replication, and opt-in parallel execution for long runs. Calibration methods follow Deville and Saerndal (1992) <doi:10.1080/01621459.1992.10475217>, and entropy balancing follows Hainmueller (2012) <doi:10.1093/pan/mpr025>.

License GPL (>= 2)

Copyright See file inst/COPYRIGHTS.

URL <https://github.com/weiansdata/WFC>

BugReports <https://github.com/weiansdata/WFC/issues>

Encoding UTF-8

Depends R (>= 3.6.0)

Imports digest, parallel, stats, utils

Roxygen list(markdown = TRUE)

LazyData true

Suggests cli, covr, generics, knitr, openxlsx, rmarkdown, survey,
testthat (>= 3.0.0)

VignetteBuilder knitr
Config/testthat/edition 3
Config/roxygen2/version 8.0.0
Repository <https://weiandata.r-universe.dev>
Date/Publication 2026-07-31 12:51:00 UTC
RemoteUrl <https://github.com/weiandata/wfc>
RemoteRef HEAD
RemoteSha 2e14ffb9884e0b4aae71a19e16cc1942f686b1de

Contents

as.data.frame.wf_quality_report	4
as.data.frame.wf_variance_result	4
as_svrepdesign	5
as_svydesign	5
plot.wf_auto_trim	6
plot.wf_blend_result	7
plot.wf_diagnostics	7
plot.wf_propensity_weights	8
plot.wf_weights	8
print.wf_attrition_weights	9
print.wf_auto_trim	9
print.wf_autoweigh_result	10
print.wf_blend_result	10
print.wf_cell_merge_plan	11
print.wf_collapse_plan	11
print.wf_design_data	12
print.wf_diagnostics	12
print.wf_impact	13
print.wf_influence	13
print.wf_ladder_draft	14
print.wf_pipeline	14
print.wf_poststrat_plan	15
print.wf_precheck	15
print.wf_quality_report	16
print.wf_replicate_weights	16
print.wf_validation	17
print.wf_variance_result	17
print.wf_weight_plan	18
print.wf_weights	18
summary.wf_poststrat_plan	19
wf_apply_collapse	19
wf_approve_plan	20
wf_assess_impact	20
wf_attach_weights	21

wf_attrition	21
wf_audit_export	22
wf_auto_trim	23
wf_autoweigh	24
wf_blend	25
wf_calibrate	26
wf_collapse_ladder	26
wf_compose	27
wf_diagnose	28
wf_dims	28
wf_execute_plan	29
wf_guided_execute	29
wf_guided_plan	30
wf_import_reference	31
wf_import_target	31
wf_influence	33
wf_pipeline	34
wf_plan_cells	34
wf_plan_poststrat	35
wf_plan_weights	36
wf_poststrat	37
wf_precheck	37
wf_prepare_design	38
wf_propensity	40
wf_rake	40
wf_replicates	41
wf_report	42
wf_run	43
wf_suggest_collapse	44
wf_suggest_ladder	45
wf_target_population	45
wf_target_propensity	47
wf_target_reference	48
wf_target_template	49
wf_validate	49
wf_variance	50
wfc-tidiers	51
wfc_example	52

```
as.data.frame.wf_quality_report
```

Convert a weighting quality report to a data frame

Description

Convert a weighting quality report to a data frame

Usage

```
## S3 method for class 'wf_quality_report'  
as.data.frame(x, ...)
```

Arguments

x	A wf_quality_report object.
...	Unused.

Value

The report's primary table.

```
as.data.frame.wf_variance_result
```

Coerce a variance result to a data frame

Description

Coerce a variance result to a data frame

Usage

```
## S3 method for class 'wf_variance_result'  
as.data.frame(x, ...)
```

Arguments

x	A wf_variance_result object.
...	Unused.

Value

The result table as a data frame.

as_svrepdesign	<i>Convert WFC replicate weights to a survey replicate design</i>
----------------	---

Description

Aligns recalibrated WFC replicate weights to analysis data by exact unit ID and creates a standard `survey::svrepdesign()` object. The returned design uses full-estimate-centered (`mse = TRUE`) variance so its estimates reproduce `wf_variance()`.

Usage

```
as_svrepdesign(r, data, id = "id", degf = NULL, ...)
```

Arguments

<code>r</code>	A <code>wf_replicate_weights</code> object.
<code>data</code>	Non-empty analysis data frame containing the units in <code>r</code> .
<code>id</code>	Unit-ID column in <code>data</code> ; matched to <code>r\$base\$id</code> .
<code>degf</code>	Optional survey design degrees of freedom.
<code>...</code>	Additional arguments passed to <code>survey::svrepdesign()</code> . WFC owns the variables, base/replicate weights, replication type and scales, combined-weight setting, and MSE setting.

Value

A standard `svyrep.design` object whose variables include the aligned `.wf_weight` base-weight column.

as_svydesign	<i>Convert WFC weights to a survey design</i>
--------------	---

Description

Joins calibrated WFC weights onto an analysis data frame by exact unit ID and creates a standard `survey::svydesign()` object. The `survey` package remains a suggested dependency.

Usage

```
as_svydesign(
  w,
  data,
  id = "id",
  ids = ~1,
  strata = NULL,
```

```

    fpc = NULL,
    nest = FALSE,
    ...
  )

```

Arguments

w	A wf_weights object.
data	Non-empty analysis data frame containing the units in w.
id	Unit-ID column in data; matched to w\$data\$id.
ids	Survey cluster formula, defaulting to independent units.
strata	Optional survey strata formula.
fpc	Optional finite-population-correction formula.
nest	Whether cluster IDs should be relabeled to nest within strata.
...	Additional arguments passed to survey::svydesign(). WFC owns weights and probs; they cannot be overridden.

Value

A standard survey.design2 object whose variables include the aligned .wf_weight column.

plot.wf_auto_trim	<i>Plot an automatic trim frontier</i>
-------------------	--

Description

Draws the worst design effect and residual margin error for every feasible finite cap.

Usage

```

## S3 method for class 'wf_auto_trim'
plot(x, lang = NULL, ...)

```

Arguments

x	A wf_auto_trim object.
lang	Output language.
...	Additional arguments passed to the initial plot calls.

Value

Invisibly returns x.

plot.wf_blend_result *Plot blend lambda sensitivity*

Description

Plot blend lambda sensitivity

Usage

```
## S3 method for class 'wf_blend_result'  
plot(x, lang = NULL, ...)
```

Arguments

x	A wf_blend_result containing sensitivity output.
lang	Output language.
...	Additional arguments passed to the initial plot.

Value

Invisibly returns x.

plot.wf_diagnostics *Plot weight diagnostics*

Description

Draws design effects and effective-sample-size shares by group.

Usage

```
## S3 method for class 'wf_diagnostics'  
plot(x, lang = NULL, ...)
```

Arguments

x	A wf_diagnostics object.
lang	Output language.
...	Additional arguments passed to <code>graphics::dotchart()</code> .

Value

Invisibly returns x.

plot.wf_propensity_weights

Plot propensity overlap and covariate balance

Description

Plot propensity overlap and covariate balance

Usage

```
## S3 method for class 'wf_propensity_weights'
plot(x, lang = NULL, ...)
```

Arguments

x	A wf_propensity_weights object.
lang	Output language.
...	Additional arguments passed to the initial plot calls.

Value

Invisibly returns x.

plot.wf_weights

Plot calibrated weight distributions

Description

Draws one histogram per group, including the group mean and recorded raking trim bounds when available.

Usage

```
## S3 method for class 'wf_weights'
plot(x, max_groups = 9, lang = NULL, ...)
```

Arguments

x	A wf_weights object.
max_groups	Maximum number of groups to plot.
lang	Output language.
...	Additional arguments passed to <code>graphics::hist()</code> .

Value

Invisibly returns x.

```
print.wf_attrition_weights
```

Print attrition weights

Description

Print attrition weights

Usage

```
## S3 method for class 'wf_attrition_weights'  
print(x, ...)
```

Arguments

x	A wf_attrition_weights object.
...	Unused.

Value

Invisibly returns x.

```
print.wf_auto_trim
```

Print an automatic trim recommendation

Description

Print an automatic trim recommendation

Usage

```
## S3 method for class 'wf_auto_trim'  
print(x, ...)
```

Arguments

x	A wf_auto_trim object.
...	Unused.

Value

Invisibly returns x.

```
print.wf_autoweigh_result
    Print a guided weighting result
```

Description

Print a guided weighting result

Usage

```
## S3 method for class 'wf_autoweigh_result'
print(x, ...)
```

Arguments

x	A wf_autoweigh_result object.
...	Unused.

Value

Invisibly returns x.

```
print.wf_blend_result  Print a blend result.
```

Description

Print a blend result.

Usage

```
## S3 method for class 'wf_blend_result'
print(x, ...)
```

Arguments

x	A wf_blend_result object.
...	Unused.

Value

Invisibly returns x.

```
print.wf_cell_merge_plan
```

Print an outcome-blind cell merge plan

Description

Print an outcome-blind cell merge plan

Usage

```
## S3 method for class 'wf_cell_merge_plan'
print(x, ...)
```

Arguments

x	A wf_cell_merge_plan object.
...	Reserved for future use.

Value

x, invisibly.

```
print.wf_collapse_plan
```

Print a collapse plan

Description

Print a collapse plan

Usage

```
## S3 method for class 'wf_collapse_plan'
print(x, ...)
```

Arguments

x	A wf_collapse_plan object.
...	Unused.

Value

Invisibly returns x.

print.wf_design_data *Print outcome-free survey design data*

Description

Print outcome-free survey design data

Usage

```
## S3 method for class 'wf_design_data'  
print(x, ...)
```

Arguments

x	A wf_design_data object.
...	Reserved for future use.

Value

x, invisibly.

print.wf_diagnostics *Print diagnostics*

Description

Print diagnostics

Usage

```
## S3 method for class 'wf_diagnostics'  
print(x, ...)
```

Arguments

x	A wf_diagnostics object.
...	Unused.

Value

Invisibly returns x.

print.wf_impact	<i>Print post-lock outcome impact</i>
-----------------	---------------------------------------

Description

Print post-lock outcome impact

Usage

```
## S3 method for class 'wf_impact'  
print(x, ...)
```

Arguments

x	A wf_impact object.
...	Reserved for future use.

Value

x, invisibly.

print.wf_influence	<i>Print influence diagnostics</i>
--------------------	------------------------------------

Description

Print influence diagnostics

Usage

```
## S3 method for class 'wf_influence'  
print(x, ...)
```

Arguments

x	A wf_influence object.
...	Unused.

Value

Invisibly returns x.

`print.wf_ladder_draft` *Print a collapse-ladder draft*

Description

Print a collapse-ladder draft

Usage

```
## S3 method for class 'wf_ladder_draft'  
print(x, ...)
```

Arguments

<code>x</code>	A <code>wf_ladder_draft</code> object.
<code>...</code>	Unused.

Value

Invisibly returns `x`.

`print.wf_pipeline` *Print a pipeline specification*

Description

Print a pipeline specification

Usage

```
## S3 method for class 'wf_pipeline'  
print(x, ...)
```

Arguments

<code>x</code>	A <code>wf_pipeline</code> object.
<code>...</code>	Unused.

Value

Invisibly returns `x`.

```
print.wf_poststrat_plan
```

Print a post-stratification plan

Description

Print a post-stratification plan

Usage

```
## S3 method for class 'wf_poststrat_plan'  
print(x, ...)
```

Arguments

x	A wf_poststrat_plan object.
...	Unused.

Value

Invisibly returns x.

```
print.wf_precheck
```

Print a precheck result

Description

Print a precheck result

Usage

```
## S3 method for class 'wf_precheck'  
print(x, ...)
```

Arguments

x	A wf_precheck object.
...	Unused.

Value

Invisibly returns x.

```
print.wf_quality_report
```

Print a weighting quality report

Description

Print a weighting quality report

Usage

```
## S3 method for class 'wf_quality_report'
print(x, ...)
```

Arguments

x	A wf_quality_report object.
...	Unused.

Value

Invisibly returns x.

```
print.wf_replicate_weights
```

Print replicate weights

Description

Print replicate weights

Usage

```
## S3 method for class 'wf_replicate_weights'
print(x, ...)
```

Arguments

x	A wf_replicate_weights object.
...	Unused.

Value

Invisibly returns x.

print.wf_validation	<i>Print a weight-validation result</i>
---------------------	---

Description

Print a weight-validation result

Usage

```
## S3 method for class 'wf_validation'
print(x, ...)
```

Arguments

x	A wf_validation object.
...	Unused.

Value

Invisibly returns x.

print.wf_variance_result	<i>Print a variance result</i>
--------------------------	--------------------------------

Description

Print a variance result

Usage

```
## S3 method for class 'wf_variance_result'
print(x, ...)
```

Arguments

x	A wf_variance_result object.
...	Unused.

Value

Invisibly returns x.

print.wf_weight_plan *Print a reviewable weight plan*

Description

Print a reviewable weight plan

Usage

```
## S3 method for class 'wf_weight_plan'  
print(x, ...)
```

Arguments

x	A wf_weight_plan object.
...	Reserved for future use.

Value

x, invisibly.

print.wf_weights *Print calibrated weights*

Description

Print calibrated weights

Usage

```
## S3 method for class 'wf_weights'  
print(x, ...)
```

Arguments

x	A wf_weights object.
...	Unused.

Value

Invisibly returns x.

```
summary.wf_poststrat_plan
```

Summarize a post-stratification plan

Description

Summarize a post-stratification plan

Usage

```
## S3 method for class 'wf_poststrat_plan'
summary(object, ...)
```

Arguments

object	A wf_poststrat_plan object.
...	Unused.

Value

The plan diagnostics data frame.

```
wf_apply_collapse
```

Apply a category collapse plan

Description

Applies category-merge maps consistently to sample data and a target object.

Usage

```
wf_apply_collapse(sample, target, plan)
```

Arguments

sample	Sample data frame.
target	A wf_target object.
plan	List with dim and named character vector map, or a wf_collapse_plan object from wf_suggest_collapse().

Value

A list with collapsed sample and target.

wf_approve_plan	<i>Record a human attestation for a reviewed weight plan</i>
-----------------	--

Description

Approval records are separate from plans. AI agents may prepare plans but cannot create an approval while declaring themselves as an agent.

Usage

```
wf_approve_plan(plan, approver, role, note = NULL, actor_type = "human")
```

Arguments

plan	An unchanged, review-ready wf_weight_plan.
approver	Non-empty approver name or organizational identifier.
role	Non-empty reviewing role.
note	Optional review note.
actor_type	Actor type. Only "human" is accepted.

Value

A wf_plan_approval attestation bound to the plan identity.

wf_assess_impact	<i>Assess descriptive outcome impact after weights are locked</i>
------------------	---

Description

Compares fixed unweighted and locked-weight estimates. This function cannot accept target values, selection callbacks, or planning controls, and it never recalculates weights.

Usage

```
wf_assess_impact(weights, data, id, outcomes, level = 0.95)
```

Arguments

weights	An unchanged wf_locked_weights object.
data	Analysis data frame with the exact locked unit-ID set.
id	Unique identifier column in data.
outcomes	Character vector naming outcome columns.
level	Normal-approximation confidence level.

Value

A wf_impact object with fixed descriptive summaries.

wf_attach_weights	<i>Attach locked weights to analysis data by exact unit ID</i>
-------------------	--

Description

Attach locked weights to analysis data by exact unit ID

Usage

```
wf_attach_weights(data, weights, id, weight_name = ".weight")
```

Arguments

data	Full analysis data frame.
weights	An unchanged wf_locked_weights object.
id	Unique identifier column in data.
weight_name	Name of the appended weight column.

Value

data in its original row order with one locked-weight column.

wf_attrition	<i>Estimate panel attrition weights</i>
--------------	---

Description

Models wave-to-wave retention in a panel and returns inverse-retention weights for retained units. The result inherits from wf_weights, so it can be composed with later calibration stages through [wf_compose\(\)](#).

Usage

```
wf_attrition(
  panel,
  retained,
  formula,
  id = NULL,
  by = NULL,
  stabilize = TRUE,
  trim = NULL
)
```

Arguments

panel	Full prior-wave panel data.
retained	Column indicating whether each prior-wave unit was retained into the next wave.
formula	One-sided or two-sided formula naming retention predictors.
id	Optional unique unit identifier column.
by	Optional grouping column; separate retention models are fit by group.
stabilize	Whether to multiply inverse probabilities by the group retention rate before normalization.
trim	Optional positive scalar cap, applied as $\text{trim} * \text{median}(\text{weight})$ within each group before mean-one normalization.

Value

A wf_attrition_weights object inheriting from wf_weights, with a \$balance diagnostic table and per-group retention log.

wf_audit_export	<i>Export a self-contained WFC audit file</i>
-----------------	---

Description

Writes a JSON audit record containing result provenance, pipeline metadata, optional guided-workflow decision ledger, input hashes, and user-supplied extra metadata. The writer is dependency-free and intended for machine archiving, not human formatting.

Usage

```
wf_audit_export(x, file, inputs = NULL, extra = NULL)
```

Arguments

x	A WFC result, safe workflow, plan, approval, locked weights, verified target, design-data object, or post-lock impact object.
file	Output JSON file path.
inputs	Optional named list of input objects whose hashes should be recorded.
extra	Optional additional metadata to include in the audit record.

Value

Invisibly returns file.

wf_auto_trim

*Recommend a weight-trimming cap***Description**

Sweeps candidate upper caps by rerunning raking and reports the trade-off between design effect and residual margin error. The function recommends a cap but never applies it to a production result.

Usage

```
wf_auto_trim(
  design,
  target,
  caps = c(2, 3, 4, 5, 6, 8, 10, 12),
  lo = 0.05,
  max_deff = 6,
  max_residual = 0.02,
  ...
)
```

Arguments

design	An unchanged wf_design_data.
target	An unchanged, non-demo wf_verified_target.
caps	Unique positive finite upper-cap multipliers.
lo	Positive lower-cap multiplier used for every finite candidate.
max_deff	Maximum acceptable worst-group design effect.
max_residual	Maximum acceptable worst relative margin residual.
...	Additional raking settings. ID and base-weight roles are owned by design; trim is not allowed because this function owns candidates.

Value

A wf_auto_trim object containing the candidate frontier and the recommended cap. Inf means no trimming is needed; NA means no candidate meets both criteria.

wf_autoweigh

Run a guided workflow from verified design and target objects

Description

Provides the guided WFC experience while enforcing the same WFC 2.0 input boundary used by direct calibration and approved-plan execution.

Usage

```
wf_autoweigh(
  design,
  target,
  dims,
  method = c("auto", "raking", "poststrat", "logit"),
  ladder = NULL,
  min_cell = NULL,
  bounds = c(0.3, 3),
  trim = "auto",
  max_deff = 6,
  max_residual = 0.02,
  interactive = base::interactive(),
  lang = NULL,
  ...
)
```

Arguments

design	An unchanged wf_design_data.
target	An unchanged, non-demo wf_verified_target.
dims	A wf_dims declaration matching target.
method	Calibration method. "auto" selects between supported safe engines from declared settings only.
ladder	Reviewed collapse ladder for post-stratification.
min_cell	Positive post-stratification cell-size threshold.
bounds	Two-element bounds for logit calibration.
trim	Raking trim control.
max_deff	Maximum design effect accepted by automatic trimming.
max_residual	Maximum margin residual accepted by automatic trimming.
interactive	Whether review prompts may be shown.
lang	Output language.
...	Method settings. ID and base-weight roles cannot be overridden.

Value

A wf_autoweigh_result carrying verified input identities.

wf_blend	<i>Blend online and offline calibrated estimates</i>
----------	--

Description

Combines two wf_weights sources at the estimator level. Each source is estimated within each fusion cell first; the source estimates are then combined using the effective lambda recorded in the result.

Usage

```
wf_blend(
  online,
  offline,
  by_cell,
  lambda = c("neff", "inverse_variance", "fixed"),
  lambda_fixed = NULL,
  outcome = NULL,
  level = c("cell", "group"),
  trim_lambda = c(0.05, 0.95),
  sensitivity = TRUE
)
```

Arguments

online	Online-source wf_weights.
offline	Offline-source wf_weights.
by_cell	Character vector of cell columns.
lambda	Lambda strategy: "neff", "inverse_variance", or "fixed".
lambda_fixed	Fixed lambda scalar or key table when lambda = "fixed".
outcome	Optional numeric outcome column.
level	Lambda level: "cell" or "group".
trim_lambda	Two bounds used to clamp data-driven lambdas.
sensitivity	Whether to compute a global-lambda sensitivity sweep.

Value

A wf_blend_result object.

wf_calibrate	<i>Calibrate weights from verified design and target objects</i>
--------------	--

Description

Enforces the WFC 2.0 verified-input boundary before dispatching to a calibration engine. Raw data frames and ordinary wf_target objects are not accepted.

Usage

```
wf_calibrate(design, target, method = "raking", ...)
```

Arguments

design	An unchanged wf_design_data created by wf_prepare_design() .
target	An unchanged, non-demo wf_verified_target created by wf_import_target() or wf_import_reference() .
method	Calibration method: "raking", "poststrat", "greg" (linear GREG), "logit" (bounded), "soft" (declared margin relaxation), or "ebal" (categorical entropy balancing).
...	Method settings. ID and base-weight roles come from design and cannot be overridden. Inline target moments are not supported.

Value

A wf_weights object carrying the verified input identities.

wf_collapse_ladder	<i>Declare a post-stratification collapse ladder</i>
--------------------	--

Description

Creates an ordered, cumulative, validated category-collapse ladder for post-stratification cell resolution. Level 0 is the raw classification; each named level applies one additional set of dimension maps.

Usage

```
wf_collapse_ladder(dims, ...)
```

Arguments

dims	A wf_dims object.
...	Named ladder levels, each a list of named character vectors in the form old = "new" by dimension.

Value

A wf_collapse_ladder object.

Examples

```
dims <- wf_dims(age = c("young", "old"), education = c("low", "high"))
wf_collapse_ladder(
  dims,
  level1 = list(age = c(young = "all", old = "all"))
)
```

wf_compose

*Compose multiple weighting stages***Description**

Multiplies compatible wf_weights objects into one auditable pipeline result.

Usage

```
wf_compose(..., id = NULL, normalize = c("none", "mean1", "sum"))
```

Arguments

...	Two or more wf_weights objects.
id	Optional shared ID column in each stage's \$data. If NULL, wf_compose() uses data\$id when every stage has it, or row order when no stage has an ID column.
normalize	Weight normalization mode: "none", "mean1", or "sum".

Value

A composed wf_weights object.

Examples

```
stage1 <- structure(
  list(
    data = data.frame(id = c("r1", "r2"), group = "A", weight = c(2, 3)),
    log = data.frame(group = "A"),
    achieved = NULL,
    provenance = list(method = "stage1")
  ),
  class = "wf_weights"
)
stage2 <- stage1
stage2$data$weight <- c(0.5, 2)
stage2$provenance$method <- "stage2"
wf_compose(stage1, stage2)
```

wf_diagnose	<i>Diagnose calibrated weights</i>
-------------	------------------------------------

Description

Computes per-group quality metrics for a wf_weights object.

Usage

```
wf_diagnose(w, target = NULL, sample = NULL, deff_ok = 3, deff_caveat = 10)
```

Arguments

w	A wf_weights object.
target	Optional wf_target object for residual margin checks.
sample	Reserved for future diagnostics.
deff_ok	Design-effect threshold for an OK verdict.
deff_caveat	Design-effect threshold for caveat verdict.

Value

A wf_diagnostics object.

wf_dims	<i>Declare calibration dimensions</i>
---------	---------------------------------------

Description

Declare calibration dimensions

Usage

```
wf_dims(..., .collapse = list())
```

Arguments

...	Named dimension-level pairs. Use NULL to infer levels from the target.
.collapse	Named list of collapse ladders.

Value

A wf_dims object.

Examples

```
dims <- wf_dims(gender = c("female", "male"), age = c("young", "old"))
dims
```

wf_execute_plan	<i>Execute an approved plan and lock its weights</i>
-----------------	--

Description

Revalidates the plan, human approval, design data, target evidence, and any stored cell map before calling exactly the method recorded in the plan.

Usage

```
wf_execute_plan(plan, approval, design, target)
```

Arguments

plan	An unchanged wf_weight_plan.
approval	A matching wf_plan_approval.
design	The unchanged wf_design_data used to build the plan.
target	The unchanged wf_verified_target used to build the plan.

Value

A wf_locked_weights object that also inherits from wf_weights.

wf_guided_execute	<i>Execute a guided workflow with an external human approval</i>
-------------------	--

Description

Execute a guided workflow with an external human approval

Usage

```
wf_guided_execute(workflow, approval)
```

Arguments

workflow	An unchanged wf_safe_workflow.
approval	A matching wf_plan_approval created separately.

Value

A wf_locked_weights object.

wf_guided_plan	<i>Prepare a guided safe weighting workflow</i>
----------------	---

Description

This practitioner-oriented entry point composes design preparation, verified target import, outcome-blind cell planning, and weight planning. It never approves a plan or computes weights.

Usage

```
wf_guided_plan(
  data,
  id,
  calibration,
  dims,
  target_file,
  source_file,
  source_type = c("population", "reference"),
  key_map = NULL,
  count = NULL,
  feature = NULL,
  ...
)
```

Arguments

data	Design-only data frame.
id	Unique identifier column.
calibration	Declared calibration and boundary columns.
dims	A wf_dims object.
target_file	CSV or XLSX target file.
source_file	Companion source DCF file.
source_type	Either "population" or "reference".
key_map	Population dimension-to-column mapping.
count	Population count column.
feature	Reference reciprocal-design-weight column.
...	Named safe settings: base_weight, strata, clusters, fpc, by, by_key, production, method, bounds, min_cell, max_weight_ratio, boundary, and ladder.

Value

A reviewable wf_safe_workflow; no weights are computed.

wf_import_reference	<i>Import a verified external reference-sample target</i>
---------------------	---

Description

Imports an external reference sample only when every file column has a declared design role and the companion DCF record passes the provenance and checksum checks.

Usage

```
wf_import_reference(  
  data_file,  
  source_file,  
  dims,  
  feature,  
  by = NULL,  
  production = TRUE  
)
```

Arguments

- data_file Path to a CSV or XLSX reference sample.
- source_file Path to its companion source DCF record.
- dims A wf_dims object.
- feature Name of the reciprocal-design-weight column.
- by Optional grouping variable.
- production Whether to reject demo-only sources.

Value

A wf_verified_target object.

wf_import_target	<i>Import a verified external population target</i>
------------------	---

Description

Imports CSV or Excel population margins only when a companion DCF record supplies complete provenance and a matching SHA-256 checksum.

Usage

```
wf_import_target(
  data_file,
  source_file,
  dims,
  key_map,
  count,
  by = NULL,
  by_key = NULL,
  production = TRUE
)
```

Arguments

<code>data_file</code>	Path to a CSV or XLSX population table.
<code>source_file</code>	Path to its companion source DCF record.
<code>dims</code>	A <code>wf_dims</code> object.
<code>key_map</code>	Named mapping from dimensions to data columns.
<code>count</code>	Name of the population-count column.
<code>by</code>	Optional grouping variable.
<code>by_key</code>	Optional group-key column or function.
<code>production</code>	Whether to reject demo-only sources.

Value

A `wf_verified_target` object.

Examples

```
# WFC ships a synthetic demo target beside its companion evidence record.
csv_file <- system.file(
  "extdata", "safe-target-example.csv",
  package = "WFC"
)
csv_source <- paste0(csv_file, ".source.dcf")

dims_safe <- wf_dims(sex = c("F", "M"), age = c("18-34", "35+"))

# `production = FALSE` is required here only because the bundled file is
# declared demo-only. Do not use it to admit an authoritative source whose
# evidence record is incomplete.
target <- wf_import_target(
  csv_file,
  csv_source,
  dims_safe,
  key_map = c(sex = "sex", age = "age"),
  count = "count",
  production = FALSE
)
```

```

)
target$identity

# The same demo file is refused as a production source, so a synthetic
# file cannot impersonate an authoritative population.
try(wf_import_target(
  csv_file,
  csv_source,
  dims_safe,
  key_map = c(sex = "sex", age = "age"),
  count = "count"
))

```

wf_influence

*Diagnose high-influence calibrated units***Description**

Reports the units that contribute most to weight instability. The core diagnostics are per-unit weight ratio to the group mean, share of the group's squared-weight mass, and leave-one-out design effect for the top units. When sample and target are supplied, WFC also reports the largest share of any target margin represented by that unit.

Usage

```
wf_influence(w, target = NULL, sample = NULL, id = NULL, top = 20)
```

Arguments

w	A wf_weights object with unit IDs.
target	Optional wf_target for margin-share diagnostics.
sample	Optional sample data frame aligned by id.
id	Sample ID column required when sample and target are supplied.
top	Number of highest-ratio units for which leave-one-out design effects are computed and printed.

Value

A wf_influence object with a \$table data frame.

wf_pipeline	<i>Declare a production weighting pipeline</i>
-------------	--

Description

Creates a serializable specification for a recurring weighting run. The specification records which verified target mode and calibration stage to use and which post-run validation thresholds to check.

Usage

```
wf_pipeline(target, stages, validate = NULL)
```

Arguments

target	Target declaration. Use a list with mode = "population", "reference", or "object", or pass a wf_verified_target.
stages	Named stage list with a required calibrate stage.
validate	Optional validation thresholds. Supported keys are max_deff and max_margin_dev.

Value

A wf_pipeline object.

wf_plan_cells	<i>Plan deterministic, outcome-blind support-cell merging</i>
---------------	---

Description

Uses only declared design fields, external target margins, ordered dimension levels, and an optional explicit collapse ladder. Study outcomes and custom scoring callbacks are not accepted by this interface.

Usage

```
wf_plan_cells(
  design,
  target,
  dims,
  min_cell = 5,
  max_weight_ratio = 4,
  boundary = target$by,
  ladder = NULL
)
```

Arguments

design	A wf_design_data object.
target	A non-demo wf_verified_target object.
dims	A wf_dims object.
min_cell	Minimum observed count in every retained joint design cell.
max_weight_ratio	Maximum conservative marginal target-to-design ratio.
boundary	Optional grouping column; defaults to the target grouping.
ladder	Optional explicit wf_collapse_ladder.

Value

A reviewable wf_cell_merge_plan. No weights are computed.

wf_plan_poststrat	<i>Plan post-stratification cell resolution</i>
-------------------	---

Description

Resolves each population joint cell to the finest supported collapse-ladder level without computing weights. Review this plan before running wf_poststrat().

Usage

```
wf_plan_poststrat(
  sample,
  target,
  min_cell,
  ladder,
  granularity = c("adaptive", "province"),
  empty_cell = c("redistribute", "flag", "error"),
  id = NULL
)
```

Arguments

sample	Sample data frame.
target	A wf_target object built with keep_joint = TRUE.
min_cell	Minimum sample count per resolved cell.
ladder	A wf_collapse_ladder object.
granularity	Resolution strategy, either "adaptive" or "province".
empty_cell	Empty-cell policy, one of "redistribute", "flag", or "error".
id	Reserved for future plan-level row diagnostics.

Value

A wf_poststrat_plan object.

wf_plan_weights	<i>Build a reviewable outcome-blind weight plan</i>
-----------------	---

Description

Performs structural checks and records approved-capable settings without calling a calibration engine or computing any weights.

Usage

```
wf_plan_weights(  
  design,  
  target,  
  dims,  
  method = c("raking", "logit", "poststrat"),  
  bounds = c(0.3, 3),  
  min_cell = 5,  
  cell_plan = NULL  
)
```

Arguments

design	A wf_design_data object.
target	A non-demo wf_verified_target object.
dims	A wf_dims object matching the target.
method	Planned method: "raking", "logit", or "poststrat".
bounds	Planned multiplicative lower and upper bounds, satisfying $0 < L < 1 < U$.
min_cell	Minimum support recorded in the plan.
cell_plan	Optional reviewed wf_cell_merge_plan to apply before calibration.

Value

A wf_weight_plan. The weights field is always NULL.

wf_poststrat	<i>Post-stratify verified design data to a verified joint target</i>
--------------	--

Description

Enforces the WFC 2.0 verified-input boundary before cell-level post-stratification.

Usage

```
wf_poststrat(design, target, min_cell, ladder, ...)
```

Arguments

design	An unchanged wf_design_data.
target	An unchanged, non-demo wf_verified_target with joint cells.
min_cell	Minimum sample count per resolved cell.
ladder	A wf_collapse_ladder object.
...	Post-stratification settings other than ID and base-weight roles, which are owned by design.

Value

A wf_weights object with cell_report and collapse_map.

wf_precheck	<i>Precheck sample and target compatibility</i>
-------------	---

Description

Runs structural feasibility checks before weighting.

Usage

```
wf_precheck(
  sample,
  target,
  id = NULL,
  na = c("fractional", "drop", "error"),
  max_na_dims = 2,
  thin_min = 5,
  risk_ratio = 10
)
```

Arguments

sample	Sample data frame.
target	A wf_target object.
id	Optional unique unit identifier column.
na	Missing calibration data policy.
max_na_dims	Maximum allowed missing calibration dimensions per row.
thin_min	Minimum unweighted support before warning.
risk_ratio	Target/sample share ratio warning threshold.

Value

A wf_precheck object.

Examples

```
dims <- wf_dims(gender = c("female", "male"))
pop <- data.frame(gender = c("female", "male"), count = c(55, 45))
target <- wf_target_population(pop, c(gender = "gender"), "count", dims)
sample <- data.frame(id = 1:4, gender = c("female", "male", "female", "male"))
wf_precheck(sample, target, id = "id")
```

wf_prepare_design	<i>Prepare outcome-free survey design data</i>
-------------------	--

Description

Creates a strict design object containing only identifiers, calibration variables, and declared sampling-design fields. Any undeclared column blocks construction so study outcomes cannot silently enter weight planning.

Usage

```
wf_prepare_design(
  data,
  id,
  calibration,
  base_weight = NULL,
  strata = NULL,
  clusters = NULL,
  fpc = NULL
)
```

Arguments

<code>data</code>	A data frame containing design variables only.
<code>id</code>	Name of the unique, non-missing record identifier column.
<code>calibration</code>	Names of columns permitted for calibration.
<code>base_weight</code>	Optional name of a positive base-weight column.
<code>strata</code>	Optional names of stratification columns.
<code>clusters</code>	Optional names of cluster columns.
<code>fpc</code>	Optional names of finite-population-correction columns.

Value

A `wf_design_data` object.

Examples

```
design_frame <- data.frame(
  id = sprintf("r%02d", 1:8),
  sex = rep(c("F", "M"), 4),
  age = rep(c("18-34", "35+"), each = 4),
  base_weight = 1,
  stringsAsFactors = FALSE
)

design <- wf_prepare_design(
  design_frame,
  id = "id",
  calibration = c("sex", "age"),
  base_weight = "base_weight"
)
design

# An outcome column has no declared design role, so it blocks construction
# instead of silently entering weight planning.
with_outcome <- cbind(
  design_frame,
  satisfaction = seq(40, 70, length.out = 8)
)
try(wf_prepare_design(
  with_outcome,
  id = "id",
  calibration = c("sex", "age"),
  base_weight = "base_weight"
))
```

wf_propensity	<i>Correct a non-probability sample by inverse-propensity pseudo-weighting.</i>
---------------	---

Description

Fits the membership model declared in a `wf_target_propensity()` object and converts each on-line unit's fitted membership probability into a pseudo-design weight. The result is a `wf_weights` object suitable as an `init_weight` for `wf_rake()` / `wf_poststrat()` and as a stage in `wf_compose()`.

Usage

```
wf_propensity(
  target,
  weight = c("ipw", "kernel", "matching"),
  stabilize = TRUE,
  trim = NULL
)
```

Arguments

target	A <code>wf_target_propensity</code> object.
weight	Pseudo-weight form. Only "ipw" is executable in this release; "kernel" / "matching" are reserved.
stabilize	Use stabilized IPW ($\pi_{\text{bar}} / \text{phat}$) to tame extreme weights.
trim	Optional positive scalar: clamp weights above <code>trim * median(w)</code> .

Value

A `wf_propensity_weights` object inheriting from `wf_weights`, with `$overlap` and `$balance` diagnostics.

wf_rake	<i>Rake verified design data to a verified external target</i>
---------	--

Description

Enforces the WFC 2.0 verified-input boundary before grouped raking.

Usage

```
wf_rake(design, target, ...)
```

Arguments

design	An unchanged wf_design_data.
target	An unchanged, non-demo wf_verified_target.
...	Raking settings other than ID and base-weight roles, which are owned by design.

Value

A wf_weights object carrying verified input identities.

wf_replicates	<i>Generate re-calibrated replicate weights for variance estimation.</i>
---------------	--

Description

Perturbs base weights by bootstrap, jackknife, or BRR multipliers and re-runs a calibration pipeline (refit) on each replicate, so the resulting variance captures calibration uncertainty. Pair with [wf_variance\(\)](#).

Usage

```
wf_replicates(
  data,
  refit,
  method = c("bootstrap", "jackknife", "brr"),
  R = 500,
  strata = NULL,
  clusters = NULL,
  id = NULL,
  base_weight = NULL,
  seed = NULL,
  rho = 0,
  parallel = FALSE,
  progress = FALSE
)
```

Arguments

data	Input data frame (one row per unit).
refit	A closure function(data, weights) -> wf_weights that re-runs the calibration pipeline using weights as the base/initial weights.
method	Replication method.
R	Number of bootstrap replicates (ignored for jackknife / BRR).
strata	Optional stratum column name (single stratum if NULL).
clusters	Optional PSU column name (each row is its own PSU if NULL).

id	Optional id column aligning replicate weights (row order if NULL).
base_weight	Optional starting base-weight column (all 1 if NULL).
seed	Optional integer seed for the bootstrap draws.
rho	Fay's BRR shrinkage parameter in $[0, 1)$. Used only when method = "brr"; rho = 0 gives standard BRR.
parallel	Whether to re-run replicate refits with forked parallelism where available, using at most two workers. Windows falls back to serial execution.
progress	Whether to show a cli progress bar when cli is installed.

Value

A wf_replicate_weights object.

wf_report	<i>Build a weighting quality report</i>
-----------	---

Description

Creates one structured dossier with manager and analyst projections. It can report calibrated/composed weights or a blend result and render the same payload as Markdown or dependency-free HTML.

Usage

```
wf_report(
  w,
  target = NULL,
  audience = c("manager", "analyst", "decision", "statistician"),
  lang = NULL,
  output = c("object", "markdown", "html"),
  file = NULL
)
```

Arguments

w	A wf_weights, wf_blend_result, wf_safe_workflow, or wf_impact object.
target	Optional wf_target for margin-residual diagnostics.
audience	Report projection: "manager"/"decision" for a compact decision view or "analyst"/"statistician" for full detail.
lang	Output language. Resolution follows explicit argument, options(wfc.lang), session locale, then English fallback.
output	Return a structured object, Markdown, or standalone HTML.
file	Optional output path for Markdown or HTML.

Value

A wf_quality_report for output = "object"; otherwise a rendered character string, or invisibly the structured report when file is used.

Examples

```
# Panel attrition weights are derived from the panel itself, so this
# example runs end to end. Reporting calibrated weights follows the same
# call; see vignette("safe-weighting-workflow") for that path, which
# additionally requires a verified target and a recorded approval.
panel <- data.frame(
  id = paste0("p", 1:24),
  region = rep(c("A", "B"), each = 12),
  retained = rep(c(TRUE, TRUE, TRUE, FALSE), times = 6),
  age = rep(c(25, 35, 45, 55, 65, 75), times = 4),
  sex = rep(c("female", "male"), times = 12),
  stringsAsFactors = FALSE
)
weights <- wf_attrition(
  panel,
  retained = "retained",
  formula = ~ age + sex,
  id = "id",
  by = "region"
)

wf_report(weights, audience = "decision")

analyst <- wf_report(weights, audience = "statistician")
names(analyst$sections)
```

wf_run

*Run a production weighting pipeline***Description**

Executes a wf_pipeline() specification against verified design and target objects, records provenance, and evaluates declared validation thresholds.

Usage

```
wf_run(
  spec,
  sample,
  dims = NULL,
  population = NULL,
  reference = NULL,
  base_weight = NULL
)
```

Arguments

spec	A wf_pipeline object.
sample	An unchanged wf_design_data object.
dims	Optional wf_dims used to verify target dimensions.
population	A verified population target for population mode.
reference	A verified reference target for reference mode.
base_weight	Deprecated runtime base-weight input. WFC 2.0 requires the base-weight role to be declared in sample; supplying this argument raises a safety error.

Value

A wf_weights object with pipeline provenance and optional \$pipeline_validation.

wf_suggest_collapse	<i>Suggest collapse plans from precheck findings</i>
---------------------	--

Description

Converts selected precheck issues into a reviewable collapse plan using collapse ladders declared in wf_dims().

Usage

```
wf_suggest_collapse(
  precheck,
  dims,
  checks = c("cat_infeasible", "support_thin", "risk_extreme_ratio"),
  max_steps = 1
)
```

Arguments

precheck	A wf_precheck object.
dims	A wf_dims object with optional collapse ladders.
checks	Precheck identifiers eligible for collapse suggestions.
max_steps	Maximum number of ladder steps to inspect per dimension.

Value

A wf_collapse_plan object.

wf_suggest_ladder	<i>Draft a post-stratification collapse ladder</i>
-------------------	--

Description

Examines worst-group category support and proposes adjacent category merges in the explicit level order declared by `wf_dims()`. The result is a draft for review; no input is modified and no merge is applied automatically.

Usage

```
wf_suggest_ladder(sample, target, dims, min_cell = 5)
```

Arguments

sample	Sample data frame.
target	A <code>wf_target</code> object.
dims	A <code>wf_dims</code> object with explicit ordered levels for every calibration dimension.
min_cell	Minimum required unweighted sample support in every group.

Value

A `wf_ladder_draft` containing reviewable levels and a validated `wf_collapse_ladder` in `$ladder`.

Examples

```
data(wfc_example)
target <- wf_target_population(
  wfc_example$population,
  c(gender = "gender", age = "age"),
  "count",
  wfc_example$dims,
  by = "province"
)
wf_suggest_ladder(wfc_example$sample, target, wfc_example$dims, min_cell = 25)
```

wf_target_population	<i>Target from external population data</i>
----------------------	---

Description

Converts arbitrary population data to a canonical `wf_target` object.

Usage

```

wf_target_population(
  pop,
  key_map,
  count,
  dims,
  by = NULL,
  by_key = NULL,
  scale = c("population", "sample", "custom"),
  sample = NULL,
  totals = NULL,
  keep_joint = FALSE
)

```

Arguments

pop	Population data frame.
key_map	Named character vector mapping wf_dims dimensions to population columns.
count	Population count column.
dims	A wf_dims object.
by	Optional grouping variable.
by_key	Optional group key column name or function.
scale	Target scale, one of "population", "sample", or "custom".
sample	Required when scale = "sample".
totals	Required when scale = "custom".
keep_joint	Whether to retain per-group joint population cells.

Value

A wf_target object.

Examples

```

dims <- wf_dims(gender = c("female", "male"))
pop <- data.frame(gender = c("female", "male"), count = c(55, 45))
wf_target_population(pop, c(gender = "gender"), "count", dims)

```

wf_target_propensity	<i>Build a propensity target: stacked reference frame and membership model spec.</i>
----------------------	--

Description

Stacks a self-selected online sample and a probability reference sample into one frame with a membership indicator, so the online sample's selection propensity can be modelled. No model is fit here; execution happens in [wf_propensity\(\)](#).

Usage

```
wf_target_propensity(  
  online,  
  reference,  
  formula,  
  method = c("logit", "rf", "gbm"),  
  by = NULL,  
  id = NULL  
)
```

Arguments

online	Data frame: the self-selected (non-probability) sample.
reference	Data frame: the probability reference sample.
formula	Two-sided membership formula, e.g. <code>member ~ age + edu</code> . The right-hand side names the model predictors; the left-hand side names the membership indicator the constructor creates (1 online, 0 reference).
method	Fit backend. Only "logit" is executable in this release; "rf" / "gbm" are reserved and abort in wf_propensity() .
by	Optional grouping column present in both frames; the propensity model is fit within each group.
id	Optional id column in online; when NULL, online units are identified by row order.

Value

A `wf_target_propensity` object.

wf_target_reference	<i>Target from a weighted reference sample</i>
---------------------	--

Description

Builds a canonical wf_target from a reference sample where feature is the reciprocal of the design weight.

Usage

```
wf_target_reference(  
  ref,  
  feature,  
  dims,  
  by = NULL,  
  feature_na = c("error", "drop"),  
  feature_gt1 = c("warn", "allow")  
)
```

Arguments

ref	Reference sample data frame.
feature	Feature value column equal to 1 / design_weight.
dims	A wf_dims object.
by	Optional grouping variable.
feature_na	Policy for missing feature values.
feature_gt1	Policy for feature values greater than one.

Value

A wf_target object.

Examples

```
dims <- wf_dims(gender = c("female", "male"))  
ref <- data.frame(gender = c("female", "male"), feature = c(0.5, 0.25))  
wf_target_reference(ref, "feature", dims)
```

wf_target_template	<i>Create a safe external-target import template</i>
--------------------	--

Description

Writes a CSV or Excel data template and a separate companion DCF source record. The source filename appends .source.dcf to the complete data filename so CSV and Excel files cannot accidentally share a checksum.

Usage

```
wf_target_template(file, dims, by = NULL, example = FALSE)
```

Arguments

file	Output path ending in .csv or .xlsx.
dims	A wf_dims object with declared levels when example = TRUE.
by	Optional grouping-column name.
example	Whether to write synthetic, demo-only rows instead of a blank template.

Value

Invisibly, a list with data_file and source_file paths.

wf_validate	<i>Compare calibrated weights against a reference release</i>
-------------	---

Description

Detects drift in recurring production weights by comparing group coverage, design effect, effective sample size, total weight, optional margin residuals, and per-unit weight ratios for matching IDs.

Usage

```
wf_validate(
  new,
  reference,
  target = NULL,
  max_deff_delta = 1,
  max_ess_loss = 0.2,
  max_total_shift = 0.05,
  max_margin_delta = 0.01,
  max_ratio_p99 = 2,
  on_issue = c("warn", "error", "none")
)
```

Arguments

<code>new</code>	New <code>wf_weights</code> object.
<code>reference</code>	Reference <code>wf_weights</code> object.
<code>target</code>	Optional <code>wf_target</code> used to compare margin residual drift.
<code>max_deff_delta</code>	Maximum allowed increase in design effect.
<code>max_ess_loss</code>	Maximum allowed fractional ESS loss.
<code>max_total_shift</code>	Maximum allowed fractional group-total shift.
<code>max_margin_delta</code>	Maximum allowed absolute margin-residual increase.
<code>max_ratio_p99</code>	Maximum allowed normalized 99th percentile unit-weight ratio; the reciprocal threshold is applied to the 1st percentile.
<code>on_issue</code>	Reaction when drift is detected: "warn", "error", or "none".

Value

A `wf_validation` object.

<code>wf_variance</code>	<i>Combine replicate weights and an estimator into a variance and CI.</i>
--------------------------	---

Description

Applies the unified replication rule $\text{Var} = \text{scale} * \sum_r \text{rscales}_r * (\text{theta}_r - \text{theta})^2$ to any estimator, using the `(scale, rscales)` stored by `wf_replicates()`.

Usage

```
wf_variance(
  replicates,
  estimator,
  data,
  level = 0.95,
  ci = c("normal", "percentile")
)
```

Arguments

<code>replicates</code>	A <code>wf_replicate_weights</code> object.
<code>estimator</code>	A closure function(<code>weights, data</code>) -> numeric (scalar or named vector).
<code>data</code>	The data frame the estimator reads.
<code>level</code>	Confidence level in (0, 1).
<code>ci</code>	Interval type: "normal" or (bootstrap only) "percentile".

Value

A wf_variance_result object.

wfc-tidiers

Broom-style projections for WFC results

Description

These S3 methods return base data frames and are registered for the generic names supplied by the suggested generics package. They do not require or return tibbles, and all programmatic column names remain stable English keys.

Usage

```
## S3 method for class 'wf_weights'
tidy(x, ...)

## S3 method for class 'wf_diagnostics'
tidy(x, ...)

## S3 method for class 'wf_blend_result'
tidy(x, ...)

## S3 method for class 'wf_variance_result'
tidy(x, ...)

## S3 method for class 'wf_weights'
glance(x, ...)

## S3 method for class 'wf_diagnostics'
glance(x, ...)

## S3 method for class 'wf_blend_result'
glance(x, ...)

## S3 method for class 'wf_variance_result'
glance(x, ...)

## S3 method for class 'wf_weights'
augment(x, data, id = "id", ...)
```

Arguments

x A supported WFC result object.

... Additional arguments. `augment.wf_weights()` does not accept additional arguments.

data	Analysis data to augment with unit weights.
id	Unit-ID column in data, matched to x\$data\$id.

Value

tidy() returns an object’s primary detail table. glance() returns a one-row summary. augment() returns data with .weight and .feature appended in data-row order.

wfc_example	<i>Simulated survey weighting example data</i>
-------------	--

Description

A small, fully simulated dataset for examples and tests. It contains no private source records.

Usage

wfc_example

Format

- A list with three elements:
- sample** Simulated respondent-level sample data.
 - population** Simulated population cell counts.
 - dims** A wf_dims object for gender and age.

Source

Generated by data-raw/make-wfc-example.R.

Index

* datasets

wfc_example, 52

as.data.frame.wf_quality_report, 4
as.data.frame.wf_variance_result, 4
as_svrepdesign, 5
as_svydesign, 5
augment.wf_weights(wfc-tidiers), 51

glance.wf_blend_result(wfc-tidiers), 51
glance.wf_diagnostics(wfc-tidiers), 51
glance.wf_variance_result(wfc-tidiers), 51
glance.wf_weights(wfc-tidiers), 51
graphics::dotchart(), 7
graphics::hist(), 8

plot.wf_auto_trim, 6
plot.wf_blend_result, 7
plot.wf_diagnostics, 7
plot.wf_propensity_weights, 8
plot.wf_weights, 8
print.wf_attrition_weights, 9
print.wf_auto_trim, 9
print.wf_autoweigh_result, 10
print.wf_blend_result, 10
print.wf_cell_merge_plan, 11
print.wf_collapse_plan, 11
print.wf_design_data, 12
print.wf_diagnostics, 12
print.wf_impact, 13
print.wf_influence, 13
print.wf_ladder_draft, 14
print.wf_pipeline, 14
print.wf_poststrat_plan, 15
print.wf_precheck, 15
print.wf_quality_report, 16
print.wf_replicate_weights, 16
print.wf_validation, 17
print.wf_variance_result, 17

print.wf_weight_plan, 18
print.wf_weights, 18

summary.wf_poststrat_plan, 19

tidy.wf_blend_result(wfc-tidiers), 51
tidy.wf_diagnostics(wfc-tidiers), 51
tidy.wf_variance_result(wfc-tidiers), 51
tidy.wf_weights(wfc-tidiers), 51

wf_apply_collapse, 19
wf_approve_plan, 20
wf_assess_impact, 20
wf_attach_weights, 21
wf_attrition, 21
wf_audit_export, 22
wf_auto_trim, 23
wf_autoweigh, 24
wf_blend, 25
wf_calibrate, 26
wf_collapse_ladder, 26
wf_compose, 27
wf_compose(), 21, 40
wf_diagnose, 28
wf_dims, 28
wf_dims(), 45
wf_execute_plan, 29
wf_guided_execute, 29
wf_guided_plan, 30
wf_import_reference, 31
wf_import_reference(), 26
wf_import_target, 31
wf_import_target(), 26
wf_influence, 33
wf_pipeline, 34
wf_plan_cells, 34
wf_plan_poststrat, 35
wf_plan_weights, 36
wf_poststrat, 37

`wf_poststrat()`, [40](#)
`wf_precheck`, [37](#)
`wf_prepare_design`, [38](#)
`wf_prepare_design()`, [26](#)
`wf_propensity`, [40](#)
`wf_propensity()`, [47](#)
`wf_rake`, [40](#)
`wf_rake()`, [40](#)
`wf_replicates`, [41](#)
`wf_replicates()`, [50](#)
`wf_report`, [42](#)
`wf_run`, [43](#)
`wf_suggest_collapse`, [44](#)
`wf_suggest_ladder`, [45](#)
`wf_target_population`, [45](#)
`wf_target_propensity`, [47](#)
`wf_target_propensity()`, [40](#)
`wf_target_reference`, [48](#)
`wf_target_template`, [49](#)
`wf_validate`, [49](#)
`wf_variance`, [50](#)
`wf_variance()`, [5](#), [41](#)
`wfc-tidiers`, [51](#)
`wfc_example`, [52](#)